

Copyright
by
Patrick John Casey
2009

**Real-Time Estimation of MIG Welding Weld Bead
Width using an IR Camera**

by

Patrick John Casey, B.S.

THESIS

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2009

**Real-Time Estimation of MIG Welding Weld Bead
Width using an IR Camera**

APPROVED BY

SUPERVISING COMMITTEE:

Joseph J. Beaman Jr., Supervisor

David Bourell

Dedicated to my wonderful family and to all my amazing friends who might
as well be family.

Acknowledgments

I would like to thank Dr. Joseph Beaman for being everything I could ask for in a thesis advisor. Without his caring guidance and encouragement it would not have been possible. I would also like to thank the Office of Naval Research (grant numbers N00014-07-1-1133 and N00014-09-1-0585) for their support of this project because their funding is what made this research feasible. I would also like to thank everyone in my office including Mike Cholette, Cameron Booth, Tim Silverman, Marcus Musselman and Jeff Krohl because the culture of teaching and learning is what equipped me with the tools possible to perform research. Finally, I would like to thank my family and friends who made me the man I am today.

Real-Time Estimation of MIG Welding Weld Bead Width using an IR Camera

Patrick John Casey, M.S.E.
The University of Texas at Austin, 2009

Supervisor: Joseph J. Beaman Jr.

Current manufacturing process controls are principally based only on statistical performance. The next evolution is to make physics based models combined with the state of the art sensors and actuators to control the manufacturing processes. In this paper, metal inert gas welding is used as an example of how the first steps in developing a reliable estimation technique to implement a physics based controller. The weld bead geometry will be the main focus because it is crucial to creating a quality weld. This paper uses an IR camera to generate and evaluate multiple weld bead width estimation techniques and characterizes their corresponding standard deviations. Also a Gaussian Mixture Model (GMM) is used to fit the temperature linescan data to fit an analytical function to the numerical data. The GMM is then used to estimate the weld bead width. Finally, the optimal linescan location is calculated to produce the best possible weld bead estimation. The result is that only one of the estimation techniques actually follows a step input and

the optimal linescan location is 4 mm from the back of the arc. Furthermore, the GMM provides an excellent fit to the temperature linescan, but does not increase the accuracy of the estimate.

Table of Contents

Acknowledgments	v
Abstract	vi
List of Tables	x
List of Figures	xi
Chapter 1. Introduction	1
Chapter 2. Background	4
2.1 Metal Inert Gas Welding	4
2.1.1 MIG Welding Bead Geometry	6
2.1.2 Welding Defects	6
2.2 Temperature Readings	7
2.3 State of the Art Welding Estimation and Control	9
Chapter 3. Weld Bead Width Estimations	11
3.1 First Derivative Absolute Maxima and Minima Estimate . . .	11
3.2 Edge Inflection Estimate	13
3.3 Peak Inflection Estimate	15
3.4 Gaussian Mixture Model Fit (GMM)	17
Chapter 4. Experimental Setup	21
4.1 Welding Test Station	21
4.2 Metal Inert Gas Welding Parameters	22
4.3 Infrared Camera	23

Chapter 5. Results	26
5.1 Estimate Error	26
5.2 Bias Correction	29
5.3 Weld Bead Step Change	32
5.4 Gaussian Mixture Model Fit Estimates	34
5.5 Optimal Linescan Location	36
Chapter 6. Conclusion	39
Appendices	41
Appendix A. Weld Bead Width LabVIEW Programs	42
A.1 First Derivative Estimate LabVIEW Code	42
A.2 Edge Inflection Estimate LabVIEW Code	43
A.3 Peak Inflection Estimate LabVIEW Code	45
Appendix B. Weld Bead Width Estimation using Gaussian Mixture Modeling	47
B.1 Matlab Code for Generating a GMM Fit	47
B.2 Weld Bead Width Estimation using GMM Fits	59
Bibliography	61
Vita	62

List of Tables

2.1	Table of Common Steel Emissivities	8
-----	----------------------------------------------	---

List of Figures

2.1	A schematic of the MIG welding process	5
2.2	A schematic of the weld bead geometry	6
3.1	A graphical representation of where the first derivative estimation is taken	11
3.2	A graphical representation of where the edge inflection estimation is taken	14
3.3	A graphical representation of where the peak inflection estimation is taken	16
3.4	A sample temperature linescan from the IR Camera	18
3.5	A graphical representation of how the Gaussian Mixture Model uses a Gaussian basis functions to generate a curve fit	19
3.6	Sample data fit by a Gaussian Mixture Model	20
4.1	Welding Test Station	21
4.2	A picture of the IR Camera used to collect temperature data	24
4.3	A sample of where the linescan is taken on the weld specimen	25
5.1	The average estimate errors along the length of the weld	27
5.2	The standard deviation of the estimate errors along the length of the weld	28
5.3	Bias correction for all the estimates	30
5.4	The estimate bias repeatability based on the variation of the bias from test to test	31
5.5	The actual weld bead for the step experiment	32
5.6	The estimate bias repeatability based on the variation of the bias from test to test	33
5.7	Estimate errors compared to GMM fit estimate errors	35
5.8	The estimate residuals as a function of the linescan location	37
5.9	The standard deviation of the residual as a function of the linescan location	38

Chapter 1

Introduction

Current statistical-based manufacturing process control requires a substantial number of a prior ensemble data sets in order to specify acceptable process variation limits. As a result, current process control is not usually effective in predicting product defects in short run and small lot manufacturing where the number of ensemble data sets is limited. By the time the current process controls are calibrated by data, the short run can be over. To overcome this limitation, a hybrid measurement-model system, cyber-enabled manufacturing to predict, control and prevent defects for short runs in a manufacturing process is proposed. This thesis concerns applying this proposed concept to gas-metal-arc welds. This is just a representative manufacturing process for which we have existing experimental hardware. In the course of future research, it is intended to develop algorithms and cyber-enabled manufacturing techniques that will be adaptable to a broad spectrum of manufacturing processes. The proposed control structure will be a departure from the current statistical-based process control. The implementation of the proposed method requires the ability to accurately model the physical system in order to predict its future state based on the estimated current state and input variables. This model must be simulated in less than the characteristic time scale of the pro-

cess for it to be useful in implementing control. In this vein, a portion of the research team will explore the hardware and algorithms necessary to maximize simulations while minimizing simulation time. Along with a physical model, an array of sensors will be used to provide measurements to a state observer that estimates of the state of the system.

In physics-based control systems, such as the ones used in cyber-enabled manufacturing systems, highly accurate modeling often requires a set of non-linear equations to describe the dynamics and the measurement process. These systems, like their traditional linear dynamics equations based counterparts are subject to noise. In addition, the sensors do not give perfect measurement since their signals can be corrupted with bias and noise. These uncertainties prevent gaining direct knowledge of these systems states. Several estimation methods exist for estimating the state of these types of systems, each with varying performances. As a piece of the overall project goal this paper will focus on the estimation of the weld bead width and characterize the associated noise.

Measuring and estimating the weld bead width can be critically useful and practical to both industry and researchers. Directly measuring other variables for weld quality can be difficult or impractical. Therefore, these variables must be estimated in order to monitor and control the quality of the weld. For example if an expensive single run part is welded, destructive tests are not an option and non-destructive tests do not always find all the defects. This is where weld variable estimation and control can be crucial in analyzing the

quality of the weld during the process. This means that the single run part can be implemented with greater confidence in the weld and thus a higher chance of success. The goal of this thesis is to create a real-time measurement of the weld bead width and to characterize the noise associated with the measurement. Measuring the weld bead width is a first step toward monitoring the quality of the weld. Furthermore, characterizing the noise associated with the measurement will lead to a smooth integration to an estimation and control algorithm.

Chapter 2

Background

2.1 Metal Inert Gas Welding

Metal inert gas (MIG) welding is one of the most common techniques to join metals. In particular the process of interest uses a consumable electrode that is deposited on the base metal. MIG welding uses an electrical current to generate heat which melts both the base metal and the electrode metal. The molten metal mixes in the gap between the metals and forms a fully fused piece of material. Also a shielding gas is essential to making a good weld because it reduces unwanted chemical reactions such as oxidation. Figure 2.1 is a schematic of a wire fed MIG welding process. The direction of travel indicates how the weld torch is moving with respect to the workpiece. The contact tube is a copper insert that conducts electricity to energize the electrode. The electrode is a consumable metal that generates an electric arc with the workpiece. The electrode is then deposited onto the workpiece as the weld progresses. The shielding gas is an inert gas and is sprayed around the electrode to reduce the amount of unwanted chemical reactions. This gas is piped in from an external gas tank. The molten weld metal is a combination of the melted electrode and workpiece material. The solidified weld metal is the what the molten weld metal becomes after it has had sufficient time to cool

down. Finally the workpiece is the original pieces of material, which require welding to be joined.

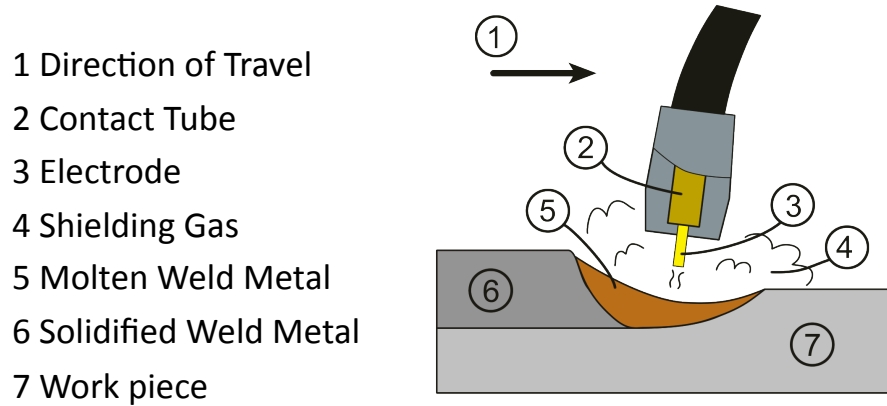


Figure 2.1: A schematic of the MIG welding process

There are many factors that influence the quality of the weld. These factors can be broadly characterized in two different categories: the welding variables and material selection. Both of these categories are equally important because if the material selection is incompatible the weld will never be strong and if the welding variables are wrong the process will also fail. The main focus of this paper is on the welding parameters because the material selection is based on metallurgical compatibility and is done completely offline. On the other hand the welding variables can be changed during the actual weld, which makes them feasible to control. The welding variables that have the greatest influence on the quality of the weld are the arc voltage, wire feed rate and specimen feed rate. The wire feed rate is how fast the consumable electrode is feed through the welding torch and the specimen feed rate is how fast the weld torch moves over the base material. MIG welding is generally done by skilled

professionals who are able to manually tune these three variables to generate a high quality weld. The ultimate goal is to reduce the amount of human error involved in the welding process, by eliminating manual control.

2.1.1 MIG Welding Bead Geometry

The quality of the weld is greatly influenced by the geometry of the bead geometry. The weld bead geometry has four major components: bead width, penetration, height and heat affected zone. All of these features are illustrated in figure 2.2.

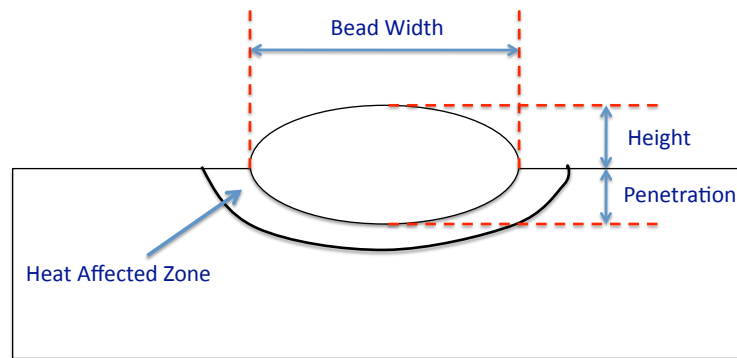


Figure 2.2: A schematic of the weld bead geometry

2.1.2 Welding Defects

When a weld does not turn out correctly it is because there are defects in the weld. These defects can weaken the bond and lead to premature failure. One of the major problems is transversal cracking which occurs when the filler metal undergoes too much stress during contraction as it cools. This can happen if the wrong materials are used, but also if the penetration to width

ratio is too small. This can happen if there is a large gap between the two base metals. Cracking is a catastrophic failure and which requires the weld to be completely redone. Therefore, the width to depth ratio should be readily monitored in order to maintain a suitable depth to width ratio. Another major defect with a weld can be a lack of fusion to the base material. This is where the filler metal seems to have bonded with the base metal, but has instead just been laid on top without any adhesion. This problem is very difficult to identify since it can look like a good weld, but the reality is that it is an extremely unsafe bond. This can be mitigated by having the bead better "wet" to the base metal by having a flat bead. This can be accomplished with a high weld bead width to height ratio. Lack of fusion can be caused when the travel speed is too low and the arc is allowed to be behind the leading edge of the weld pool. This causes the weld pool to reduce the amount of heat input into the base material. If the base material receives too little heat then it will not melt and therefore not bond with the filler metal, which results in a lack of fusion defect. In conclusion, the weld bead width needs to be monitored because it is a vital component in reducing defects in the weld and creating high quality welds.

2.2 Temperature Readings

The infrared camera uses the radiation emitted from an object to calculate the temperature. The emissivity of the material is used to scale the calculated temperature to the actual value. The emissivity of a material is

the measured by the ratio of energy a given object emits verses a black body source at the same temperature. A black body source is an idealized object which emits the maximum possible amount of energy. Therefore, any real object will have an emissivity smaller than one. A table of common emissivities of metals is included in table 2.1. These emissivities show that the amount of energy emitted is drastically affected by the finish and phase of the material. For example the polished mild steel has 12.5 percent the emissivity as oxidized steel.

Table 2.1: Table of Common Steel Emissivities

Material	Temperature (°C)	Emissivity
Cold Rolled Steel	93	.75-.85
Polished Mild Steel	24	.10
Smooth Mild Steel	24	.12
Liquid Mild Steel	1599-1793	.28
Unoxidized Steel	100	.08
Oxidized Steel	25	.80

The user inputs the desired emissivity value into the IR camera to generate a calibrated temperature reading. Unfortunately, determining the correct emissivity of an object is not always an easy task. As the table shows, the finish or coating on the material greatly impacts the emissivity and consequently the temperature readings. Emissivity is also dependent on temperature, which makes an accurate calculation even more elusive. This problem can also become even more prevalent when there is a phase change in the object. For the sample process of MIG welding, there is inherently always a phase change in the welding process because it is necessary for solidification in welding to

occur. Therefore, this problem will always exist when using infrared cameras to measure the temperature of a weld.

2.3 State of the Art Welding Estimation and Control

Menaka et al. [2] developed a method for directly sensing the weld bead width. This can be accomplished by using an IR camera to measure the temperature profile of the weld specimen. A linescan measurement is the basis for the experiments. The linescan is a single row of temperatures selected from the entire temperature matrix. There is a point of inflection in the linescan that indicates the weld bead width. The reason there is an inflection point is because the emissivity of the metal varies based on the phase. The weld pool is defined by where the phase change occurs. Furthermore, the IR camera uses a constant emissivity to calculate the temperature from the thermal radiation, which skews the temperature calculation where the emissivity deviates from the set-point. This means that the temperature reading will change simply by a variation in emissivity even if the actual temperature remains constant. This variation in emissivity will cause an inflection point in the linescan reading where the weld bead is located. Therefore, the linescan can be used to directly measure the weld bead width.

Chen et al. [5] used an IR camera to estimate the depth of penetration. An isotherm around the weld pool was chosen, because the depth of penetration varies with the amount of heat input as well as the distribution of the heat. The isotherm was then fitted with an ellipse using the least squares

method. The depth of penetration and the characteristics of the ellipse were then analyzed and compared. Chen found that the area of the ellipse and the minor axis both varied with the depth of penetration.

Venkatraman et al. [1] used a similar technique as Chen, but calculated the depth of penetration differently. The peak of the linescan and the area under the linescan both indicated the depth of penetration, but the peak temperature had a linear relationship with the depth of penetration.

S. Nagarajan et al. [4] used an IR camera to measure the asymmetry of the thermal profile. Seam tracking was accomplished since a perfectly aligned weld produced a perfectly symmetrical thermal profile. This occurs because the rate of heat transfer was impeded by the interface of the two pieces of metal. Therefore, the amount of asymmetry was measured and correlated directly to the alignment of the torch.

Banerjee et al. [3] monitored various process disruptions based on the changes in the temperature gradients of the weld pool. Fluctuations of the shielding gas and the plate thickness as well as minor element contamination were observable in the temperature gradient.

Chapter 3

Weld Bead Width Estimations

3.1 First Derivative Absolute Maxima and Minima Estimate

The initial estimate technique is called the first derivative absolute maxima and minima. This estimate uses the numerical first derivative of the temperature to calculate the weld bead width. The estimated weld bead width is the distance between the absolute maxima and minima of the first derivative. Figure 3.1 is a sample linescan and a graphical representation of how the first derivative absolute maxima and minima estimate is calculated.

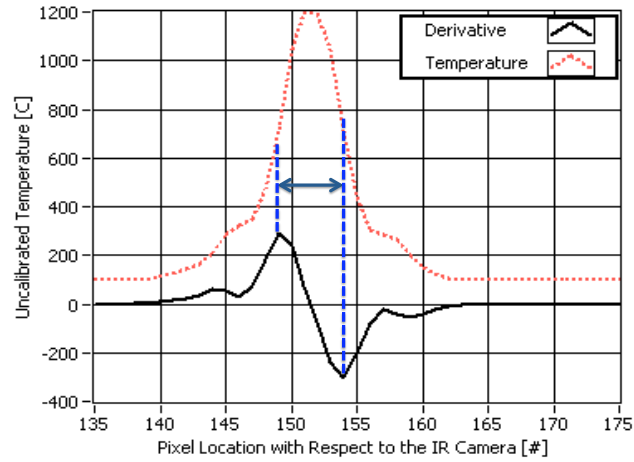


Figure 3.1: A graphical representation of where the first derivative estimation is taken

LabVIEW is used to actually calculate the estimate. The program uses a finite difference method to calculate the first derivative of the temperature. The equation for this calculation is included in equation 3.1, where T , T' and x are the temperature, the first derivative of the temperature and the pixel location respectively.

$$T'(x) = \frac{T(x) - T(x - 1)}{dx} \quad (3.1)$$

The LabVIEW program for the first derivative estimate inputs the modified IR camera data, which is a matrix of temperatures, and outputs the estimated weld bead width. The first step is to extract only the desired linescan from the temperature matrix. This will reduce the 2D data set to a 1D array. Next the program eliminates any obvious outliers from the estimation. This is done by specifying a minimum temperature threshold which must be attained before the program will consider a location to be hot enough to contain molten metal. This helps to eliminate possible numerical anomalies that cannot possibly signify the edge of the weld bead. The next part of the program calculates the numerical first derivative in the manner previously discussed. Subsequently, the program searches for the index corresponding to the absolute maximum and minimum of the first derivative. No interpolation is necessary because the absolute maximum and minimum will always occur at an exact pixel location due to the properties inherent to the finite difference method. The distance between the maximum and minimum is then converted from pixels to inches using a user defined conversion rate. This conversion

rate is experimentally calculated from the IR camera image as discussed in chapter 4. The complete block diagram for the LabVIEW program is included in appendix A.1.

3.2 Edge Inflection Estimate

The next estimate technique is the called the edge inflection estimate. This estimate uses the numerical second derivative of the temperature to calculate the weld bead width. The second derivative is used to locate inflection points of the temperature linescan. The weld bead width is calculated by the distance between the two inflection points closest to either edge of the temperature profile. Figure 3.2 shows a sample linescan and a graphical representation of how the edge inflection estimate is calculated.

LabVIEW is used to actually calculate the estimate. The program uses a finite difference method to calculate the second derivative of the temperature. The equation for this calculation is included in equation 3.2, where where T' , T'' and x are the first derivative of the temperature, the second derivative of the temperature and the pixel location respectively. This calculation is done after the first derivative is calculated from equation 3.1.

$$T''(x) = \frac{T'(x) - T'(x - 1)}{dx} \quad (3.2)$$

The LabVIEW program for the edge inflection estimate inputs the modified IR camera data, which is a matrix of temperatures, and outputs the esti-

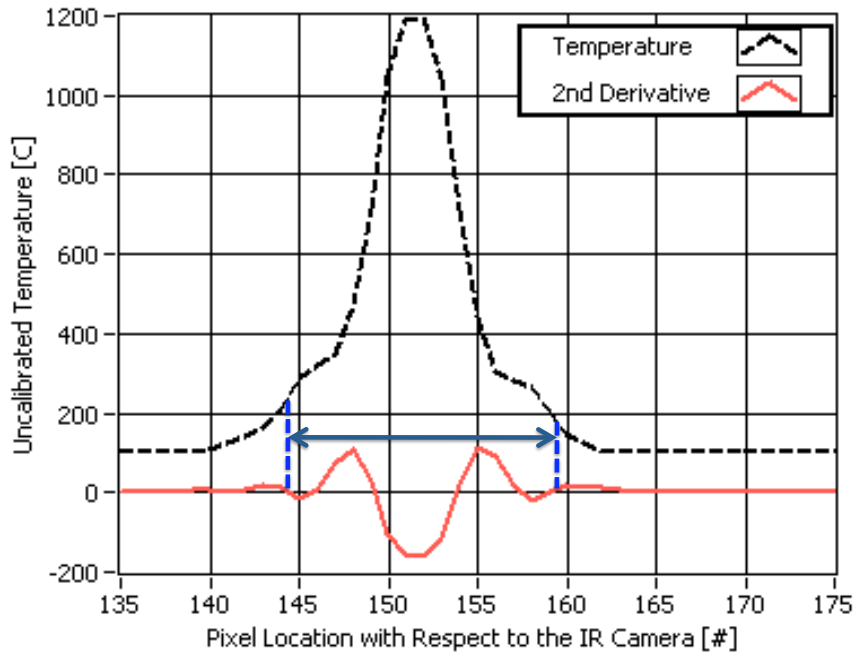


Figure 3.2: A graphical representation of where the edge inflection estimation is taken

mated weld bead width. The first step is to extract only the desired linescan from the temperature matrix. This will reduce the 2D data set to a 1D array. Then the data are split to analyze the linescan from both sides of the temperature profile. This is done by reversing the order of the array. Next the program eliminates any obvious outliers from the estimation. This is done by specifying a minimum temperature threshold which must be attained before the program will consider a location to be hot enough to contain molten metal. This helps to eliminate possible numerical anomalies that cannot possibly signify the edge of the weld bead. The next part of the program calculates the numerical first and second derivatives using equations 3.1 and 3.2 respectively.

Subsequently, the program searches for the index corresponding to the first inflection point after the threshold temperature. Linear interpolation is employed to find a more accurate inflection point location. The interpolation algorithm in LabVIEW can only find rising edges in the data. Therefore, the second derivative must be inverted to find falling edges as well. The interpolated distance between the two inflection points is then converted from pixels to inches using a user defined conversion rate. This conversion rate is experimentally calculated from the IR camera image as discussed in chapter 4. The complete block diagram for the LabVIEW program is included in appendix A.2.

3.3 Peak Inflection Estimate

The last estimate technique is the called the peak inflection estimate. This estimate also uses the numerical second derivative, from equation 3.2, of the temperature to calculate the weld bead width. The second derivative is still used to locate inflection points of the temperature linescan, but the peak inflection estimate uses a different search algorithm than the edge inflection estimate. The weld bead width is calculated by the distance between the two inflection points closest to peak of the temperature profile. Figure 3.3 shows a sample linescan and a graphical representation of how the peak inflection estimate is calculated. This estimate is a natural extension of the edge inflection technique.

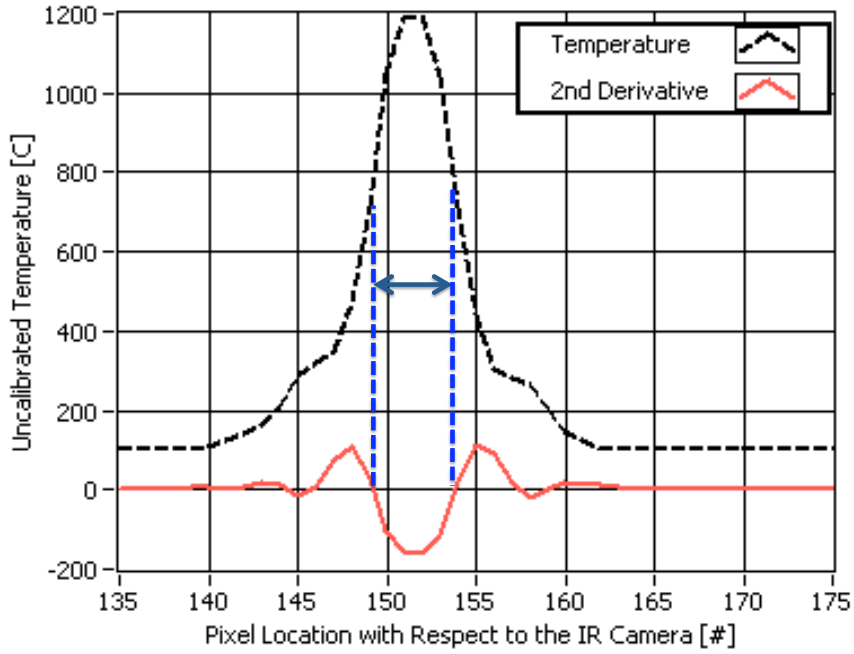


Figure 3.3: A graphical representation of where the peak inflection estimation is taken

The LabVIEW program for the peak inflection estimate inputs the modified IR camera data, which is a matrix of temperatures, and outputs the estimated weld bead width. The first step is to extract only the desired linescan from the temperature matrix. This will reduce the 2D data set to a 1D array. Then the data is split to analyze the linescan from both sides of the temperature profile. This is done by reversing the order of the array. The next part of the program calculates the numerical first and second derivatives using equations 3.1 and 3.2 respectively. Then the program finds the index of the peak temperature of the linescan, which will provide a starting point for the inflection point search algorithm. Subsequently, the program searches

for the index corresponding to the first inflection point on either side of the peak. Linear interpolation is employed to find a more accurate inflection point location. The interpolation algorithm in LabVIEW can only find rising edges in the data. Therefore, the second derivative must be inverted to find falling edges as well. The interpolated distance between the two inflection points is then converted from pixels to inches using a user defined conversion rate. This conversion rate is experimentally calculated from the IR camera image as discussed in chapter 4. The complete block diagram for the LabVIEW program is included in appendix A.3.

3.4 Gaussian Mixture Model Fit (GMM)

Noise is one of the many problems inherent with numerical data analysis. Fitting a function to match the data as closely as possible is often used to mitigate noise issues. The noise is drastically reduced because the data is converted from numerical data set to an analytical function. This eliminates the amplification of the noise inherent in numerical differentiation as well as eliminates any nonphysical spikes in the data. There are a wide range of techniques to fit a function to data, but the main difference between each fit is their basis functions. In the MIG welding linescan case there are multiple curve fits that will closely match the data set, but some require significantly more orders of magnitude to attain the same objective. By inspection the best basis function to fit the temperature linescan is a Gaussian distribution. This is because it only requires a third order of magnitude to achieve a good

fit. This is indicated by the three distinct peaks that characterize the linescan profile as shown in figure 3.4.

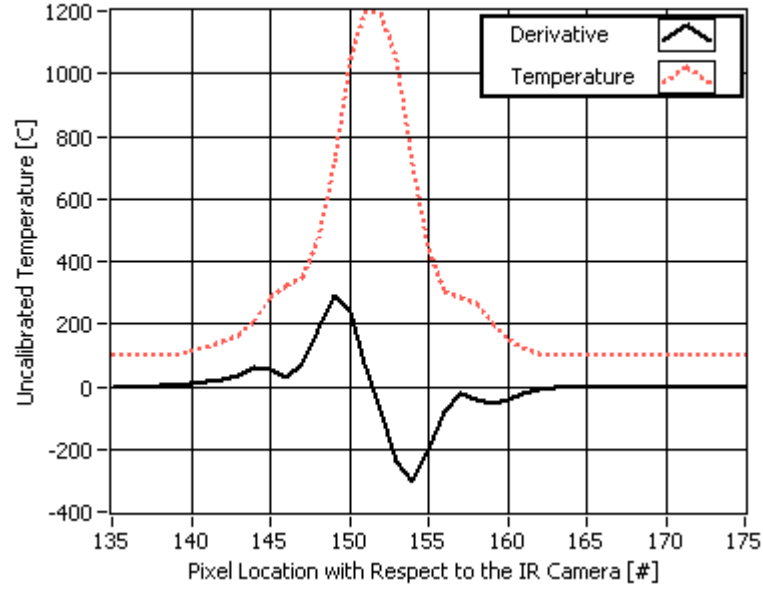


Figure 3.4: A sample temperature linescan from the IR Camera

Three independent Gaussian curves are combined linearly to generate an analytical function to fit the data with a Gaussian Mixture Model. Figure 3.5 shows how the the Gaussian curves can be linearly combined to generate an analytical Gaussian Mixture Model.

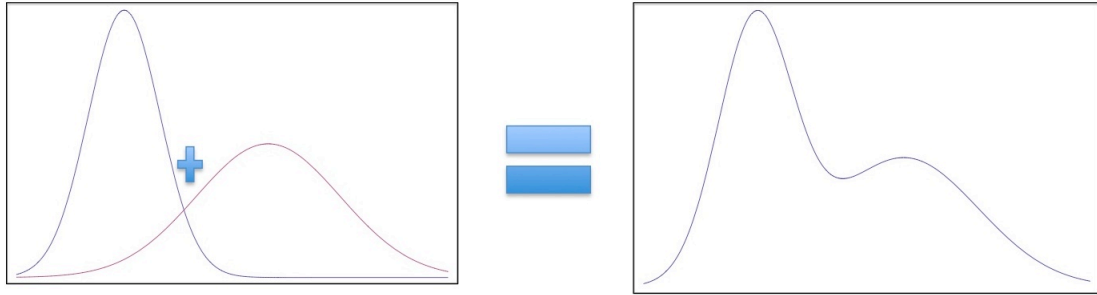


Figure 3.5: A graphical representation of how the Gaussian Mixture Model uses a Gaussian basis functions to generate a curve fit

The next step is to fit a GMM to actual data. This is done by feeding the temperature matrix from the IR camera into a Matlab program, which will generate a fit for the data. This Matlab program is included in appendix B.1. The fit is very good because it only requires a third order of magnitude and generates an accurate fit as shown by figure 3.4 which is a GMM fit to actual data. The quality of the GMM fit is characterized by visual inspection.

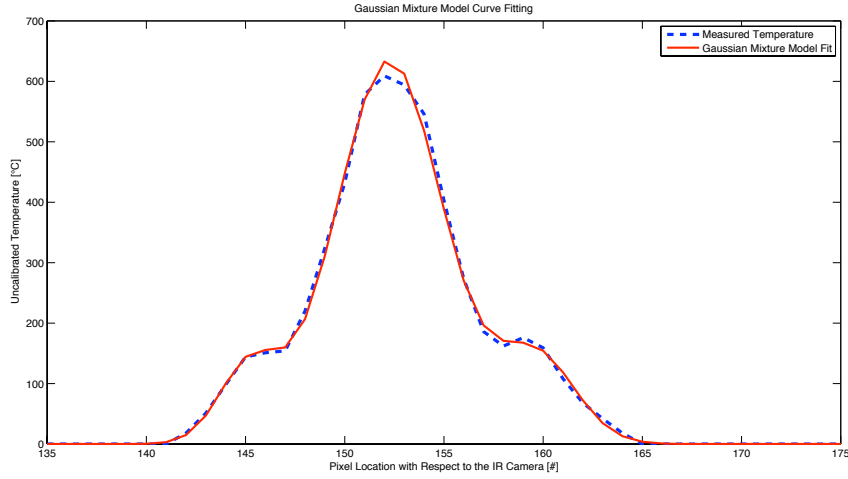


Figure 3.6: Sample data fit by a Gaussian Mixture Model

The Matlab program inputs the modified IR camera data, which is a temperature matrix, and outputs a new set of data points that correspond to the GMM fit for an individual linescan. Each GMM fit needs to be visually verified individually to ensure that the best possible fit is attained. The GMM Matlab code can generate multiple fits to the data because the code uses an algorithm that finds local solution minimums instead of global minimums. Therefore, multiple local minimums can produce multiple solutions and the best way to verify the fit is to inspect each one visually. Once the new set of data points is generated they can be used to generate new weld bead width estimates. These new estimates are done using another LabVIEW program that is included in appendix B.2. This code uses the previous estimation techniques to generate weld bead widths, but the input is now the GMM fit data instead of the raw IR camera data.

Chapter 4

Experimental Setup

4.1 Welding Test Station

The weld test station is specifically built for welding experiments. Therefore, the test station can be easily reconfigured for different experiment configurations. The main idea of the test station is to keep the welding torch fixed while moving the specimen. Figure 4.1 is a photo of the welding test station.

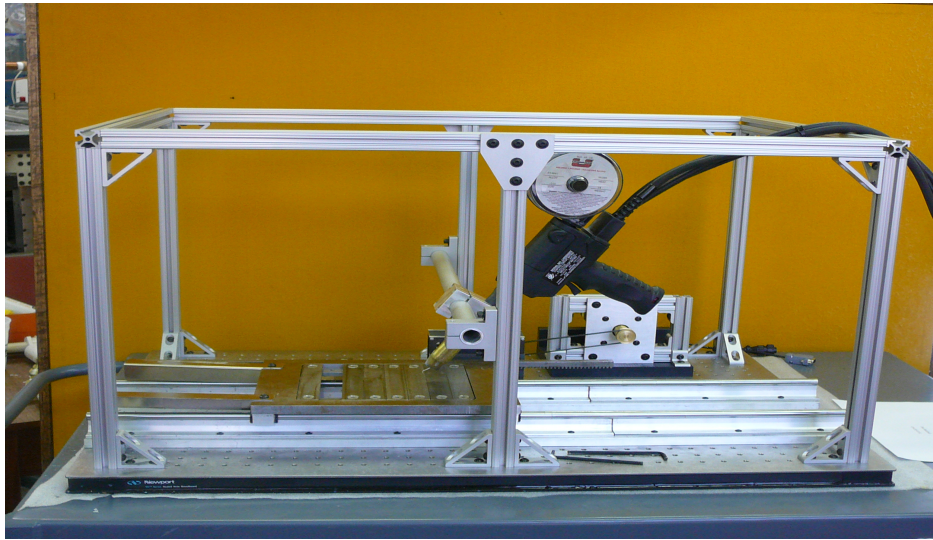


Figure 4.1: Welding Test Station

The frame for the tests station is made from 8020 brand T-Slotted

aluminum, which allows the IR camera and any other sensors to be easily positioned. The carriage holding the weld specimen moves under the torch is driven by a stepper motor which is controlled by a LabVIEW program created by MotionPlus.

4.2 Metal Inert Gas Welding Parameters

Metal inert gas welding (MIG) was done with a Millermatic 251 power source and Miller Spoolmatic 15A welding torch. All the experiments were bead on plate welds down the middle of the plate. The welding materials were 1018 low carbon steel and steel filler wire. Each plate was 0.635 cm (1/4 in) thick and 5.08 cm (2 in) wide. The surface of the plates were cleaned and scored with a wire brush to remove oxides and minimize contamination. The filler wire diameter was 0.8 mm (0.03 in) with an argon shielding gas.

All of the tests were a single pass weld with the weld specimen moving at a constant rate of 50.8 cm/min (20 in/min) and the argon gas flow rate of 12 LPM (25 CFH). For all the experiments with constant weld conditions the voltage was kept at 24 V and a wire feed rate at 10.2 m/min (400 in/min). The step test started with a voltage of 24 V and a wire feed rate of 10.2 m/min (400 in/min) and increased to voltage to 30 V and 15.2 m/min (600 in/min) to generate a step increase in the weld bead width.

The data set consisted of a total of eleven tests. Ten of the tests were of constant voltage and wire feed rate while the final test was with the step input. Ten tests were performed in order to gain statistical significance in

order to validate the mean and standard deviation calculations performed in the results section. The actual weld bead width was taken in 19 locations along each weld. These measurements were taken in the exact same location for each test. Calipers with a resolution of 0.0254 mm (0.001 in) were used to both measure the location and width of the actual weld bead width.

4.3 Infrared Camera

A Flir A320G infrared camera was used to measure the thermal radiation of the weld. The camera can be electronically calibrated to have a temperature range of -20°C to 120°C, 0°C to 350°C and 100°C to 2000°C. The camera has an accuracy of $\pm 2^\circ\text{C}$ or $\pm 2\%$ over the entire temperature range and a thermal sensitivity of less than 0.07°C (0.14°F) at 30°C (86°F). The camera has a 25° x 18.8° field of view with an array size of 320 x 240 pixels with a feature to automatically focus the camera. The spectral range of the camera is 7.5 to 13 μm with a focal plane array, uncooled microbolometer detector. A GigE ethernet connection is used to collect the image at a maximum sampling rate of 60 Hz, which is streamed in real-time from the IR camera. Each scan of the camera transfers a stream of discrete JPEG images to create a video. Each JPEG image can individually be converted to a temperature field using either LabVIEW or the Researcher Pro software created by Flir. The camera was placed in front of the torch and elevated on the test station frame. This position allowed the camera to capture the entire weld pool in front of the torch, while avoiding damage from sparks and spatter. A picture

of the IR camera that was used for all the tests is included in figure 4.2.



Figure 4.2: A picture of the IR Camera used to collect temperature data

The camera outputs temperatures as a function of pixels. This needs to be converted into millimeters in order to be useful for validating the estimate experiments. A known length is placed on the measurement test bed and the number of pixels required to span the known length gives an accurate conversion rate. For the weld bead width estimate experiments the conversion rate was 0.906 mm/pixel. This number is also the maximum resolution that the camera can achieve at that specific location. The location of the linescan on the actual weld is illustrated in figure 4.3 as well as some helpful nomenclature for the weld process.

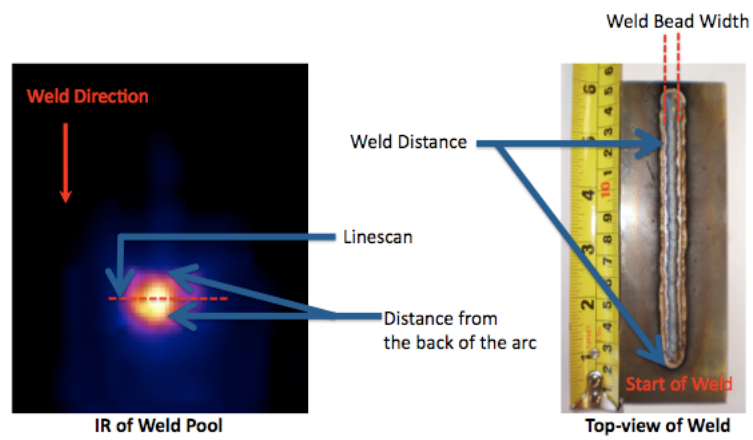


Figure 4.3: A sample of where the linescan is taken on the weld specimen

Chapter 5

Results

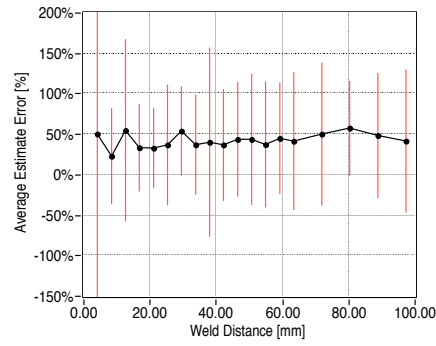
The goal of the estimate is to be as close to the actual weld bead width as possible. None of the estimates will be able to calculate the weld bead width exactly. This means that the performance of each estimate needs to be evaluated in order to determine which technique should be implemented. The most important characteristics is that when the physical system changes the estimate is able to accurately track this change. Secondly, the noise and bias of the estimate should be minimized in order to have the best estimate.

5.1 Estimate Error

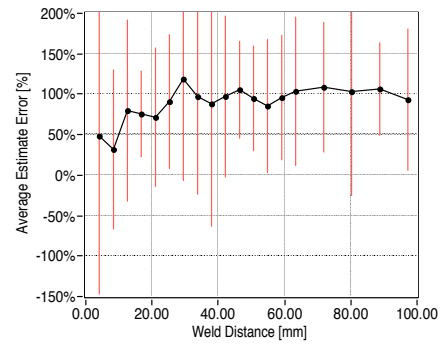
The performance of the estimates is evaluated by plotting the average estimate residual along the length of the weld. The estimate residual is calculated with equation 5.1.

$$Residual = \frac{Actual - Estimated}{Actual} \quad (5.1)$$

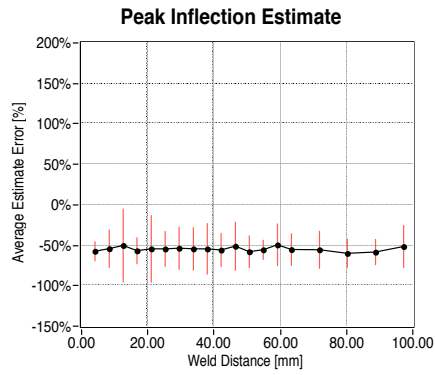
Figure 5.1 is a plot of the estimate residuals with a confidence interval of 2σ .



(a) First Derivative Estimate



(b) Edge Inflection Estimate



(c) Peak Inflection Estimate

Figure 5.1: The average estimate errors along the length of the weld

In order to fully characterize the quality of the estimate the standard deviation is graphed in figure 5.2. These standard deviations capture how the estimate residuals vary from test to test.

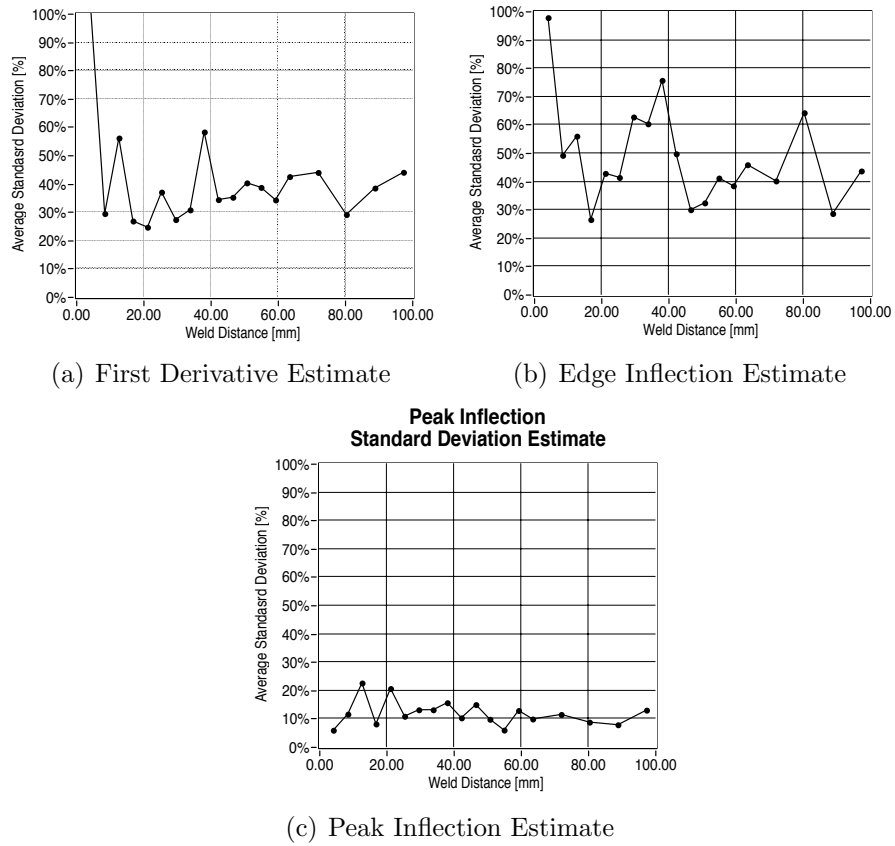
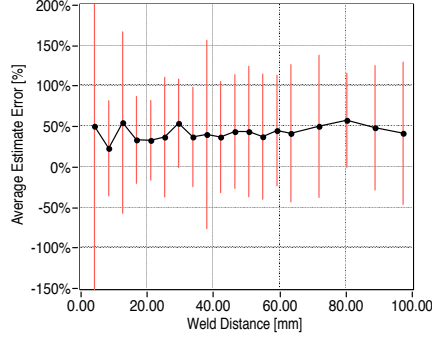


Figure 5.2: The standard deviation of the estimate errors along the length of the weld

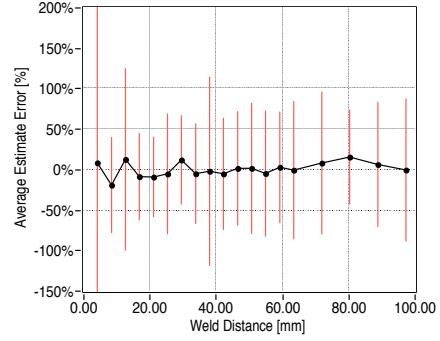
By visual analysis, the plots of residual standard deviations clearly show that the peak inflection estimate has least amount of variation among the tests. Additionally, none of the estimates seem to have a zero estimate error. Therefore, all of the estimate techniques seem to contain some biasing that will be address in the next section.

5.2 Bias Correction

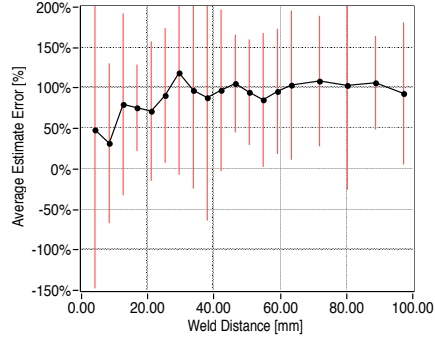
Biassing in where the mean of the estimate is not at zero. Initially this may indicate that the estimate contains a large error, but this problem can be mitigated with bias correction. Bias correction is where the mean of the estimate is subtracted away from the estimate. This will make the estimate residual centered around zero. A plot of the estimate residuals with bias correction is given in figure 5.3.



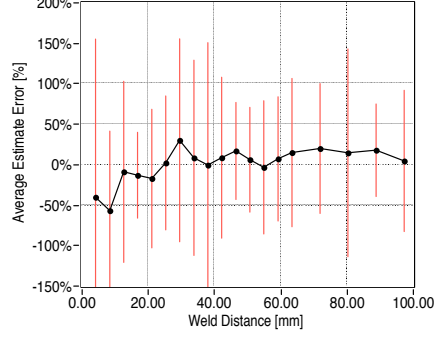
(a) First Derivative Estimate



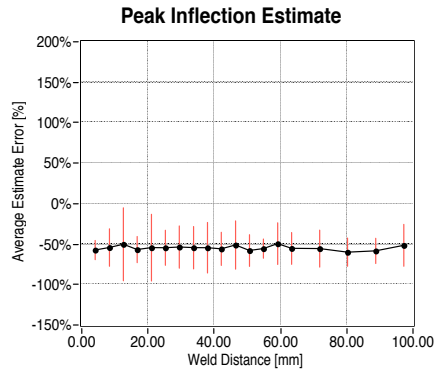
(b) Biased First Derivative Estimate



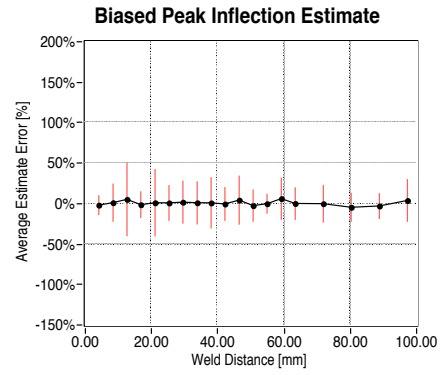
(c) Edge Inflection Estimate



(d) Biased Edge Inflection Estimate



(e) Peak Inflection Estimate



(f) Biased Peak Inflection Estimate

Figure 5.3: Bias correction for all the estimates

Now that biases have been eliminated from the estimates, the next step is to characterize how repeatable this bias correction is. This is because the bias may vary between tests, therefore the bias cannot be immediately subtracted from the estimate. Figure 5.4 show how the bias varies between tests.

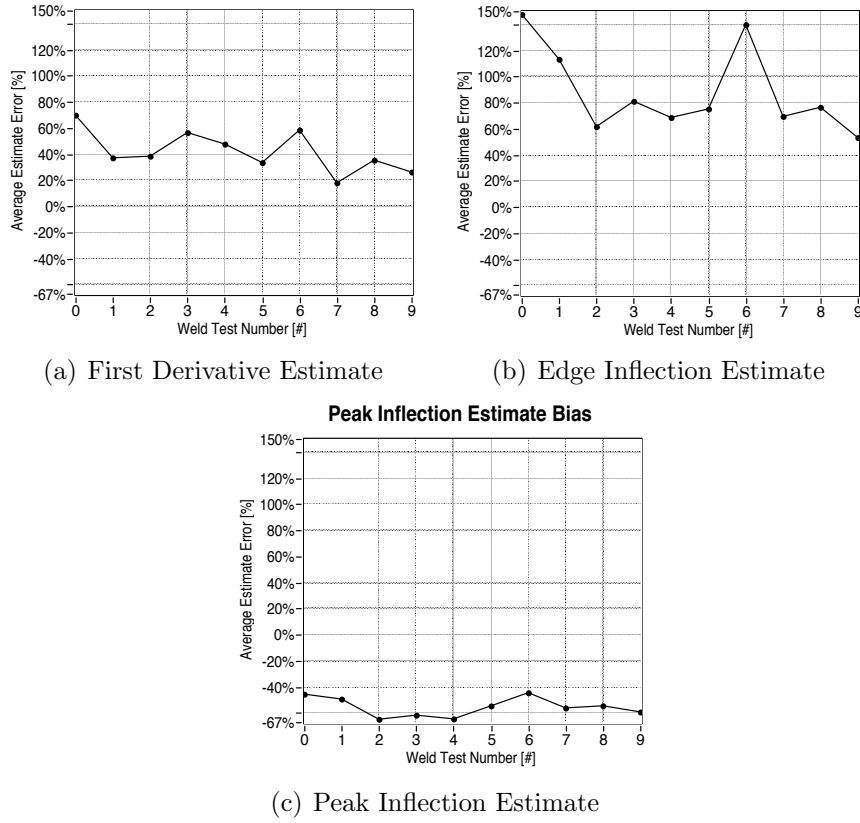


Figure 5.4: The estimate bias repeatability based on the variation of the bias from test to test

The plots in figure 5.4 show that the bias can be eliminated, but the repeatability of each estimate is different. These plots can be used to gain insight into how predictable the bias for each estimate will be. The most

obvious information contained in the bias repeatability graphs is that the peak inflection point has the most stable bias correction.

5.3 Weld Bead Step Change

Now that performance of the estimates has been characterized the next step is to determine how well it can track a step change. The step change is where the weld bead width is increased midway through the weld. The actual weld bead can be seen in figure 5.5.

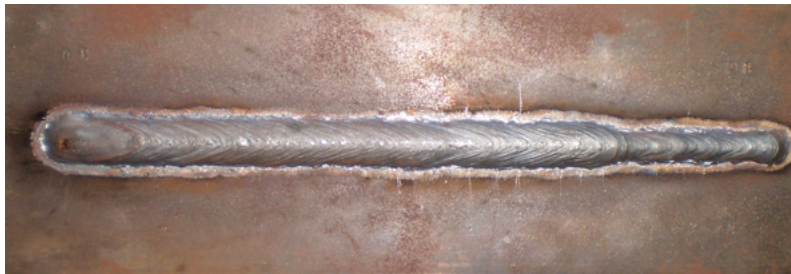


Figure 5.5: The actual weld bead for the step experiment

The raw data is plotted to verify that the estimates actually track the step change and is included in figure 5.6.

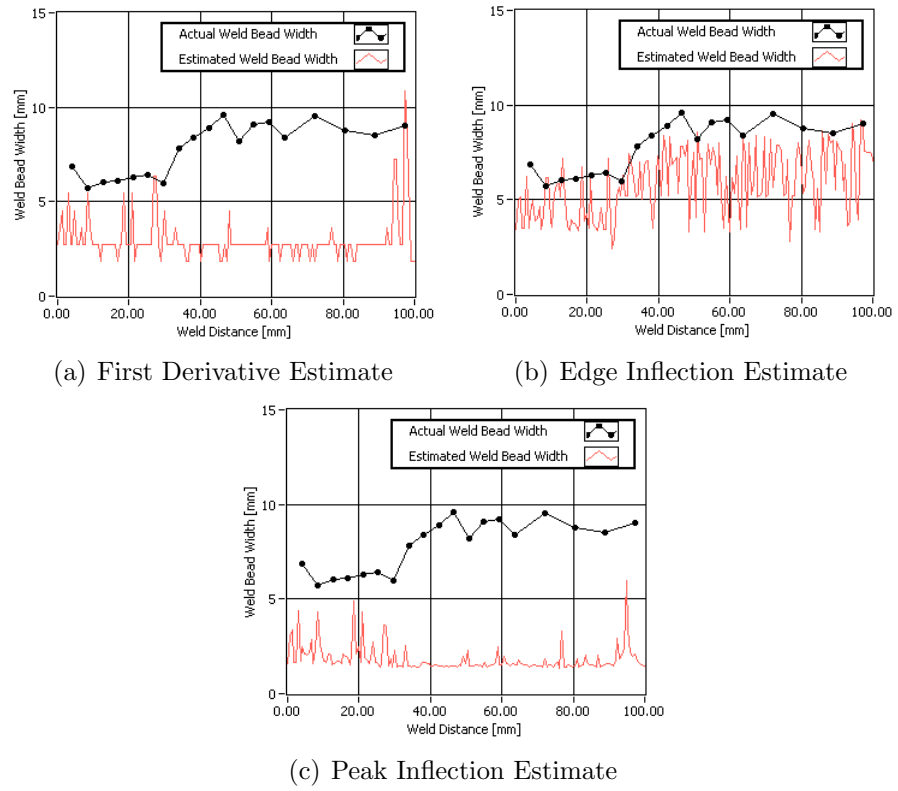
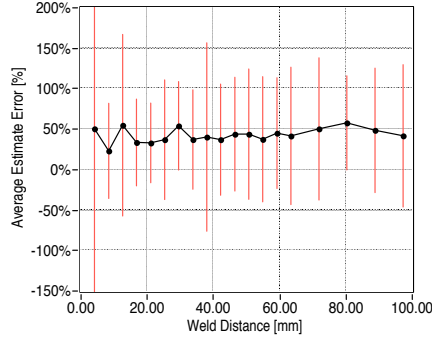


Figure 5.6: The estimate bias repeatability based on the variation of the bias from test to test

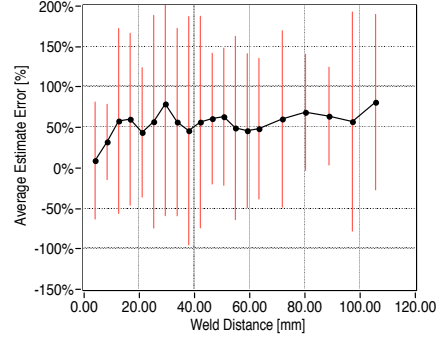
The plots above provide the most information out of any of the previous data analysis techniques. This is because they show how the estimate actually tracks the actual weld bead width. The edge inflection estimate is the only technique that actually tracks the step change. This information is invaluable, because the peak inflection estimate contains the least variation and the most bias repeatability but does not track a step change. Therefore, it cannot be used to estimate the weld bead width since it does not actual measure the physical system. Even though the edge inflection estimate is noisier it is still better because it tracks the step input.

5.4 Gaussian Mixture Model Fit Estimates

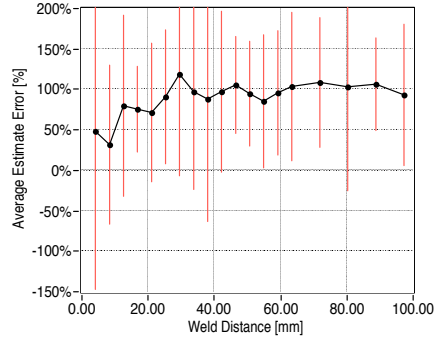
The performance of the Gaussian Mixture Model is plotted in figure 5.7.



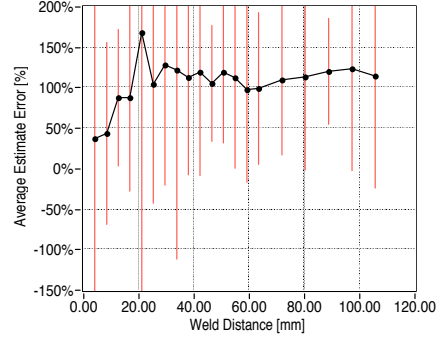
(a) First Derivative Estimate



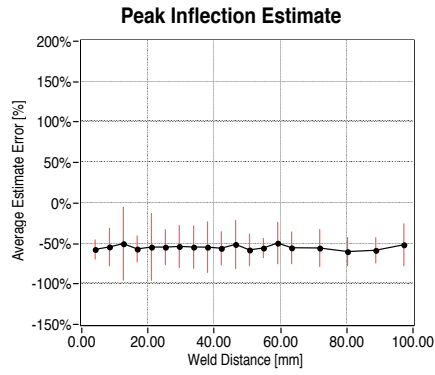
(b) Biased First Derivative Estimate



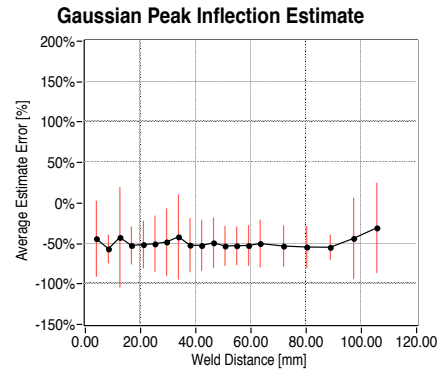
(c) Edge Inflection Estimate



(d) Biased Edge Inflection Estimate



(e) Peak Inflection Estimate



(f) Biased Peak Inflection Estimate

Figure 5.7: Estimate errors compared to GMM fit estimate errors

These plots show that the GMM does not provide any added benefit in reducing either the error or reducing the noise. Furthermore, generating the GMM requires additional processing power and time. Therefore, the calculating a GMM fit does is not warranted and should not be used to estimate the weld bead width.

5.5 Optimal Linescan Location

One major problem with estimating the weld bead width from a linescan is that the location of the linescan can change the estimate results. In order to mitigate any errors regarding the linescan location the average estimate error as a function of the distance from the back of the arc. This plot is included in figure 5.8.

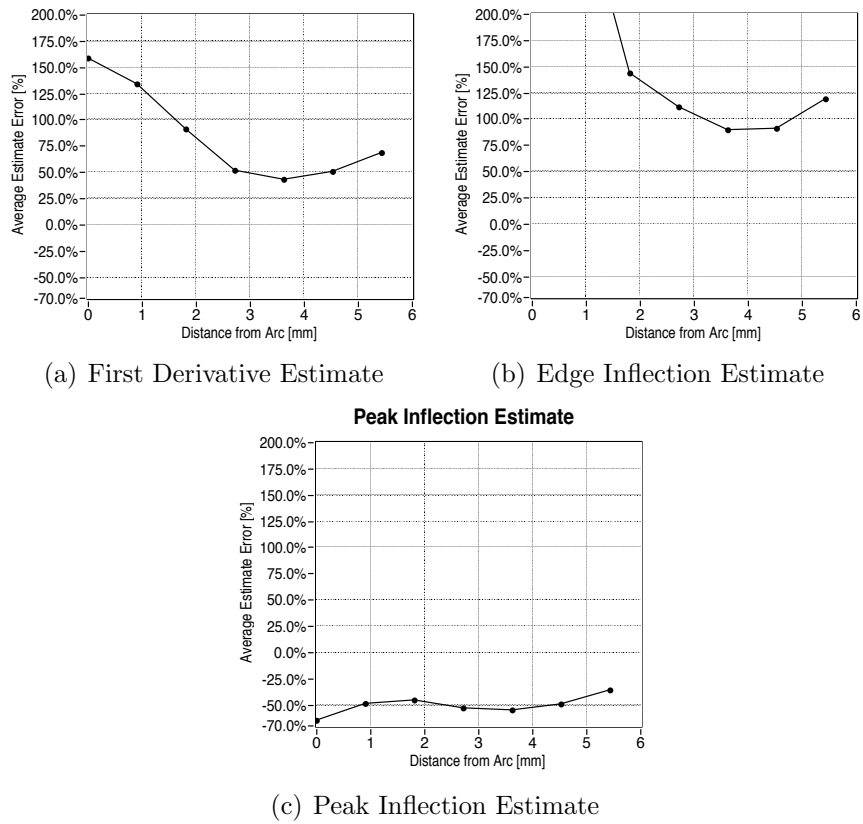


Figure 5.8: The estimate residuals as a function of the linescan location

Also the average standard deviation of the estimate error is illustrated in figure 5.9.

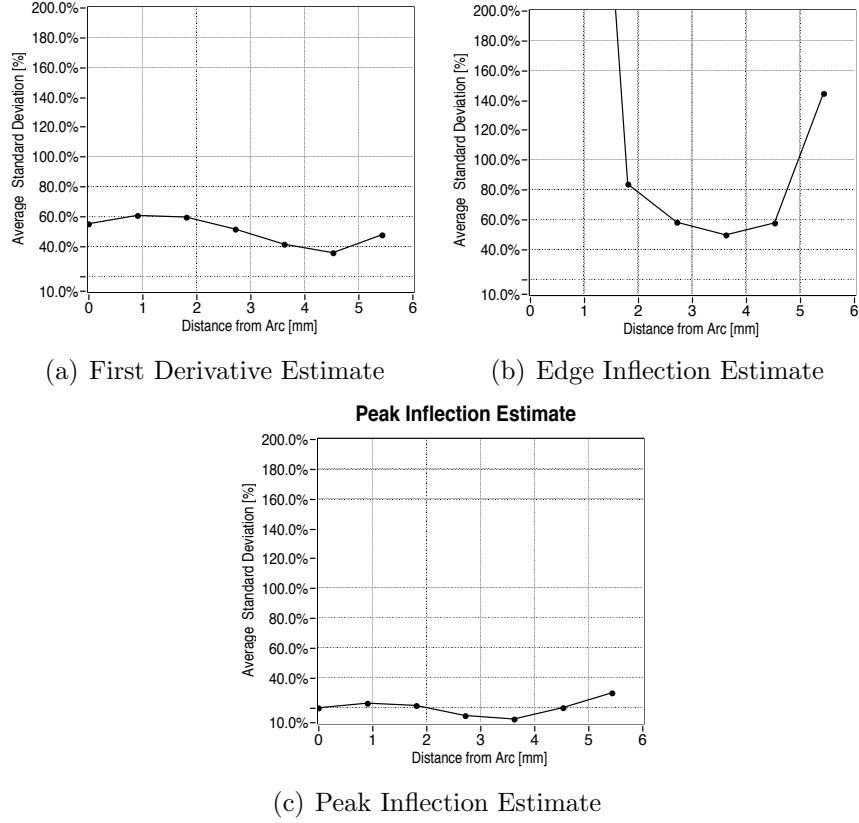


Figure 5.9: The standard deviation of the residual as a function of the linescan location

In conclusion the optimal pixel location is approximately 4 mm from the back of the arc to minimize both the average estimate error and the average standard deviation.

Chapter 6

Conclusion

In conclusion the performance of three different weld bead width estimation techniques was analyzed and evaluated. The edge inflection estimate is the only estimate that proved to be applicable because it was the only one that followed the step input. The standard deviation of each estimate was also calculated and plotted. Furthermore, bias correction was applied to the estimates to center the estimates around the origin. The repeatability of the bias was also analyzed for each of the estimates illustrating how the bias changes from test to test.

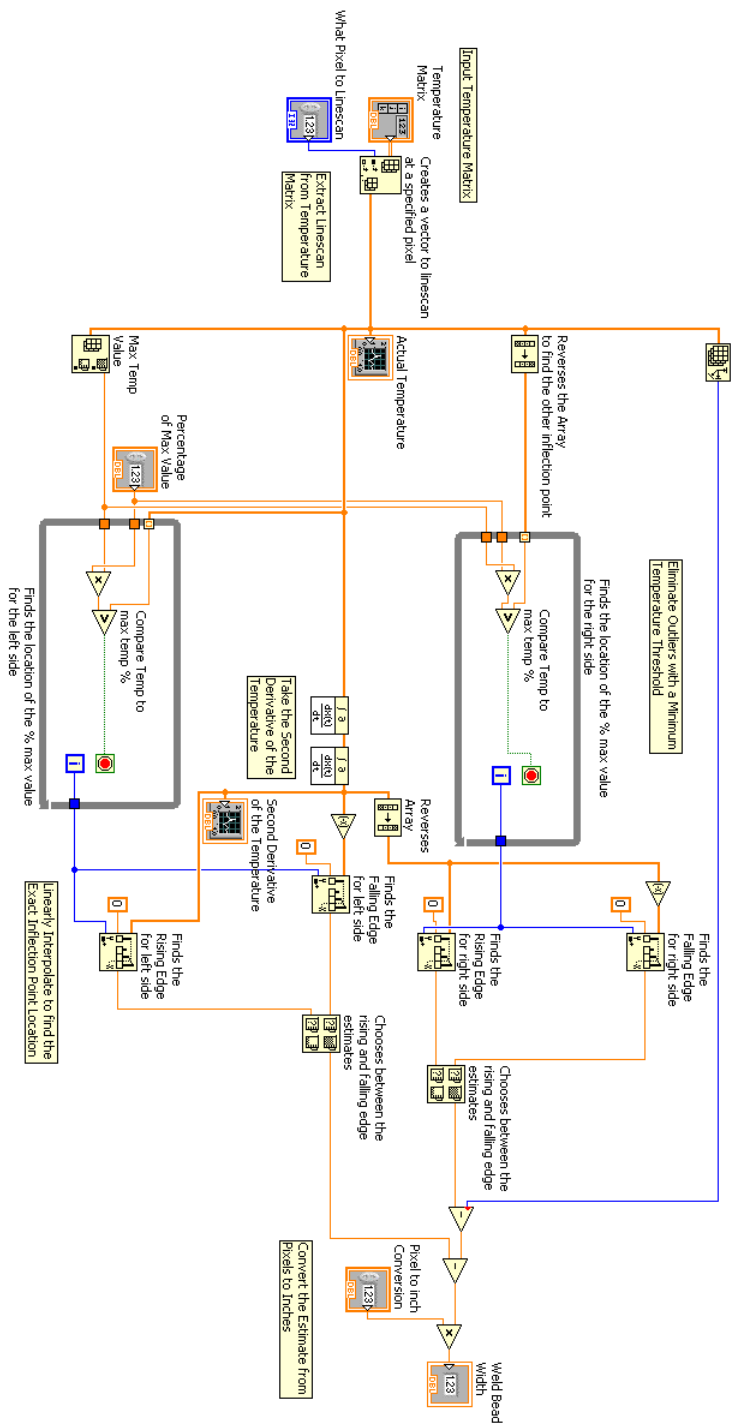
This finding is a departure from the literature because Menaka [2] performed similar analysis using both the first derivative maximum and minimum and the inflection point technique to calculate the weld bead width. Menaka found that the first derivative absolute maximum and minimum generated the best measurement of the weld bead width while still tracking a step change. The research undertaken and documented in this paper did not find the same result in testing. The main differences between the approaches is that Menaka calculated the weld bead width visually, while the research in this thesis used software to calculate everything automatically. Furthermore, Menaka also only

used a total of four points of measurement as opposed to the 19 measurements per tests with a total of 11 tests totaling 209 measurements. Therefore, the statistical significance of the measurements in this thesis is much higher. Finally, the exact location of where Menaka performed the linescan is never specified, except for being ahead of the torch head. This prompted the analysis of the optimal linescan location.

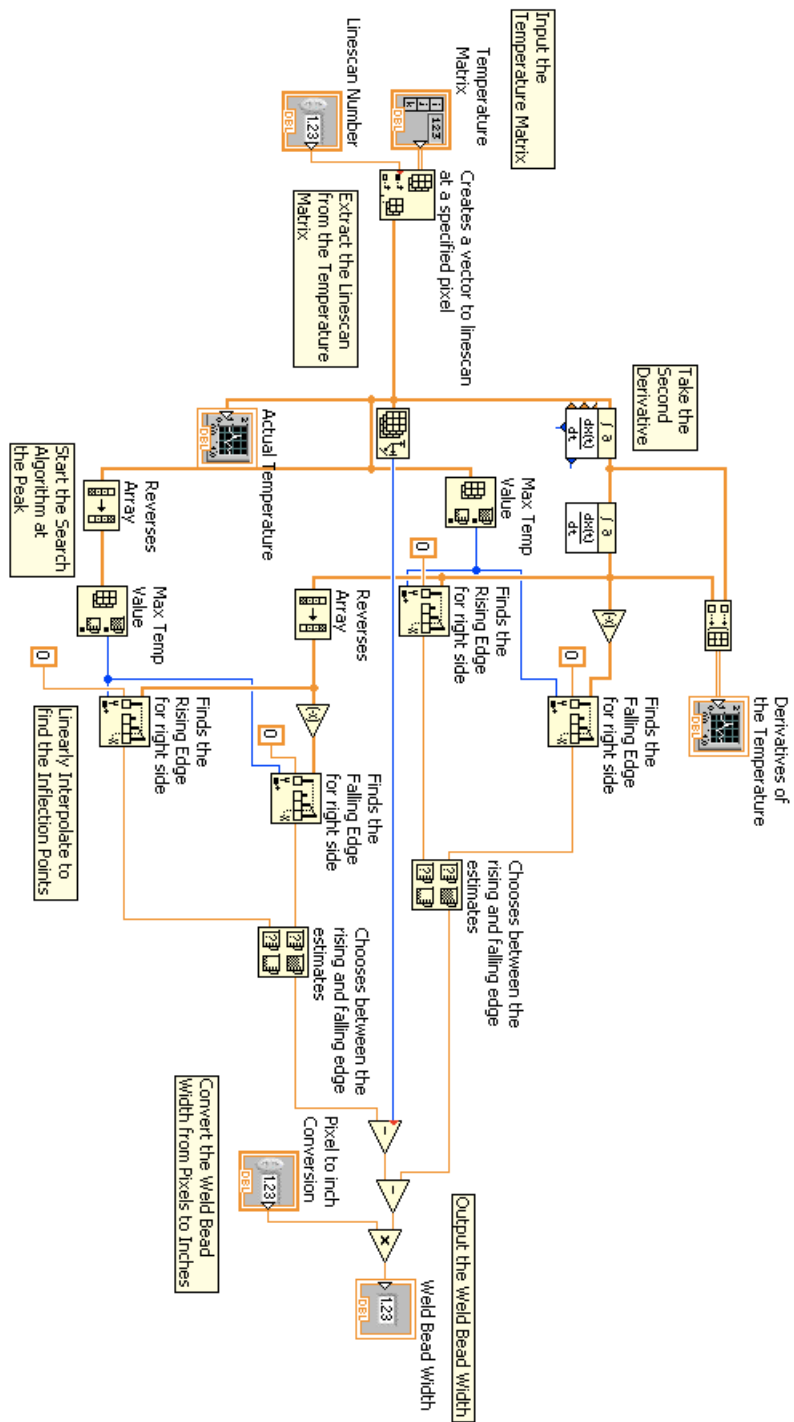
Also, the data was fit by a Gaussian Mixture Model function, which has Gaussian curves as a basis function. The Gaussian Mixture Model fit was an excellent choice of functions to fit to the data because not only did it follow the data well, but it also only had a small order of magnitude, three. The end result of the GMM curve fitting is that it did not result in a more accurate weld bead width estimation. Therefore, the GMM fit is not a helpful technique because it requires additional computing power and yields no extra benefit. The final step was to find where the optimal linescan location was to generate the most accurate estimates. The optimal linescan location was approximately 4 mm from the back of the arc. For future work, edge inflection estimation technique can now be implemented in a cyber-enabled control system because it has been proven to track the weld bead width and its standard deviation has been fully characterized.

Appendices

A.2 Edge Inflection Estimate LabVIEW Code



A.3 Peak Inflection Estimate LabVIEW Code



Appendix B

Weld Bead Width Estimation using Gaussian Mixture Modeling

B.1 Matlab Code for Generating a GMM Fit

```
function [pdf] = fitPJGMM(Measurements,Tests,LineStart,LineEnd)

    for n = 0:Measurements
        for j = 0:Tests
            Output = [];
            for k = LineStart:LineEnd
                UserInput = 0;
                while UserInput == 0

                    TestNumber = [num2str(j),'-',num2str(n),'.csv'];
                    pixel = k + csvread('Actual_Weld_Bead_Widths.csv',23,j+..
                        2,[23,j+2,23,j+2]);

                    dat = csvread(TestNumber);
                    dat = dat(pixel,:);
                    xDat = 1:length(dat);
                    dat = dat - 100;
                    % subtract off lower saturation of the IR camera

                    [pdf,scale] = GMMTempFit(xDat,dat,3);

                    % test the GMM
                    alpha = pdf.mix_coefs;
                    mu = pdf.mean;
                    sig = sqrt(pdf.covar);
                    pixels = xDat;

                    y = alpha(1)*normpdf(pixels,mu(1),sig(1)) + alpha(2)*...
                        normpdf(pixels,mu(2),sig(2)) +...
                        alpha(3)*normpdf(pixels,mu(3),sig(3));

                    %Check the Fit
                    figure
                    plot(dat,':')
                    hold on
                    plot(y.*scale,'r')
                    xlim([135 170]);

                    UserInput = input('Press 9 to accept fit +..
                        and 0 to create another fit');
                    close;

                end
            end
        end
    end

    function [pdf,scale] = GMMTempFit(xData,tempData,num_mixtures)
    % PDF = GSONGAUSSMIX(XDATA,TEMPDATA,NUM_MIXTURES,NO)

    newData = [];
    for i = 1:length(xData)
        newData = [newData, xData(i).*ones(1,(tempData(i)))];
    end
end
```

```

end
scale = sum((tempData));

M = num_mixtures;
e = newData';
options = [ 0
            0.0001
            0.0001
            0.0000
            1
            0
            0
            0
            0
            0
            0
            0
            0
            0
            0.0000
            0.1000
            0 ]';
%options(14) = 100; % maximum number of iterations
mixTemp = gmm(1, M, 'diag');
mixModel = []; % initialize mixture model cell array
pdf.num_mixture = M;
pdf.mix_coefs = zeros(M,1);
pdf.mean = zeros(M,1);
pdf.covar = zeros(M,1);

% estimation of regional pdfs for modeling errors
mix = gmminit(mixTemp, e,options); % initialize mixture model
mixmodel = gmmem(mix,e,options); % uses expectation maximization algorithm to estimate mix model
pdf.mix_coefs = mixmodel.priors';
pdf.mean = reshape(mixmodel.centres',M,1);
pdf.covar = reshape(mixmodel.covars',M,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Begin Function Definitions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mix = gmm(dim, ncentres, covar_type, ppca_dim)
%GMM Creates a Gaussian mixture model with specified architecture.
%
% Description
% MIX = GMM(DIM, NCENTRES, COVARTYPE) takes the dimension of the space
% DIM, the number of centres in the mixture model and the type of the
% mixture model, and returns a data structure MIX. The mixture model
% type defines the covariance structure of each component Gaussian:
% 'spherical' = single variance parameter for each component: stored as a vector
% 'diag' = diagonal matrix for each component: stored as rows of a matrix
% 'full' = full matrix for each component: stored as 3d array
% 'ppca' = probabilistic PCA: stored as principal components (in a 3d array
% and associated variances and off-subspace noise
% MIX = GMM(DIM, NCENTRES, COVARTYPE, PPCA_DIM) also sets the
% dimension of the PPCA sub-spaces: the default value is one.
%
% The priors are initialised to equal values summing to one, and the
% covariances are all the identity matrix (or equivalent). The centres
% are initialised randomly from a zero mean unit variance Gaussian.
% This makes use of the MATLAB function RANDN and so the seed for the
% random weight initialisation can be set using RANDN('STATE', S) where
% S is the state value.
%
% The fields in MIX are
%
% type = 'gmm'
% nin = the dimension of the space
% ncentres = number of mixture components
% covartype = string for type of variance model
% priors = mixing coefficients
% centres = means of Gaussians: stored as rows of a matrix
% covars = covariances of Gaussians
% The additional fields for mixtures of PPCA are
% U = principal component subspaces
% lambda = in-space covariances: stored as rows of a matrix
% The off-subspace noise is stored in COVARS.

```

```

%
% See also
% GMPAK, GMMUNPAK, GMMSAMP, GMMINIT, GMMEM, GMACTIV, GMPOST,
% GMPROB
%

% Copyright (c) Ian T Nabney (1996-2001)

if ncentres < 1
    error('Number of centres must be greater than zero')
end

mix.type = 'gmm';
mix.nin = dim;
mix.ncentres = ncentres;

vartypes = {'spherical', 'diag', 'full', 'ppca'};

if sum(strcmp(covar_type, vartypes)) == 0
    error('Undefined covariance type')
else
    mix.covar_type = covar_type;
end

% Make default dimension of PPCA subspaces one.
if strcmp(covar_type, 'ppca')
    if nargin < 4
        ppca_dim = 1;
    end
    if ppca_dim > dim
        error('Dimension of PPCA subspaces must be less than data.')
    end
    mix.ppca_dim = ppca_dim;
end

% Initialise priors to be equal and summing to one
mix.priors = ones(1, mix.ncentres) ./ mix.ncentres;

% Initialise centres
mix.centres = randn(mix.ncentres, mix.nin);

% Initialise all the variances to unity
switch mix.covar_type

case 'spherical'
    mix.covars = ones(1, mix.ncentres);
    mix.nwts = mix.ncentres + mix.ncentres*mix.nin + mix.ncentres;
case 'diag'
    % Store diagonals of covariance matrices as rows in a matrix
    mix.covars = ones(mix.ncentres, mix.nin);
    mix.nwts = mix.ncentres + mix.ncentres*mix.nin + ...
        mix.ncentres*mix.nin;
case 'full'
    % Store covariance matrices in a row vector of matrices
    mix.covars = repmat(eye(mix.nin), [1 1 mix.ncentres]);
    mix.nwts = mix.ncentres + mix.ncentres*mix.nin + ...
        mix.ncentres*mix.nin*mix.nin;
case 'ppca'
    % This is the off-subspace noise: make it smaller than
    % lambdas
    mix.covars = 0.1*ones(1, mix.ncentres);
    % Also set aside storage for principal components and
    % associated variances
    init_space = eye(mix.nin);
    init_space = init_space(:, 1:mix.ppca_dim);
    init_space(mix.ppca_dim+1:mix.nin, :) = ...
        ones(mix.nin - mix.ppca_dim, mix.ppca_dim);
    mix.U = repmat(init_space, [1 1 mix.ncentres]);
    mix.lambda = ones(mix.ncentres, mix.ppca_dim);
    % Take account of additional parameters
    mix.nwts = mix.ncentres + mix.ncentres*mix.nin + ...
        mix.ncentres + mix.ncentres*mix.ppca_dim + ...
        mix.ncentres*mix.nin*mix.ppca_dim;
otherwise
    error(['Unknown covariance type ', mix.covar_type]);
end

```

```

% end gmm

function mix = gmminit(mix, x, options)
%GMMINIT Initialises Gaussian mixture model from data
%
% Description
% MIX = GMMINIT(MIX, X, OPTIONS) uses a dataset X to initialise the
% parameters of a Gaussian mixture model defined by the data structure
% MIX. The k-means algorithm is used to determine the centres. The
% priors are computed from the proportion of examples belonging to each
% cluster. The covariance matrices are calculated as the sample
% covariance of the points associated with (i.e. closest to) the
% corresponding centres. For a mixture of PPCA model, the PPCA
% decomposition is calculated for the points closest to a given centre.
% This initialisation can be used as the starting point for training
% the model using the EM algorithm.
%
% See also
% GMM
%
% Copyright (c) Ian T Nabney (1996-2001)

[ndata, xdim] = size(x);

% Check that inputs are consistent
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end

% Arbitrary width used if variance collapses to zero: make it 'large' so
% that centre is responsible for a reasonable number of points.
GMM_WIDTH = max(x);

% Use kmeans algorithm to set centres
options(5) = 1;
endwhile = 0;
check = zeros(size(x,1),1);
while endwhile == 0
    [mix.centres, options, post] = kmeans(mix.centres, x, options);
    for i = 1:size(x,1)
        ind=find(mix.centres==x(i));
        if isempty(ind)
            check(i)=1;
        else
            check(i)=0;
        end
    end
    indcheck = find(check==0);
    if isempty(indcheck)
        endwhile = 1;
    end
end

% Set priors depending on number of points in each cluster
cluster_sizes = max(sum(post, 1), 1); % Make sure that no prior is zero
mix.priors = cluster_sizes/sum(cluster_sizes); % Normalise priors

switch mix.covar_type
case 'spherical'
    if mix.ncentres > 1
        % Determine widths as distance to nearest centre
        % (or a constant if this is zero)
        cdist = dist2(mix.centres, mix.centres);
        cdist = cdist + diag(ones(mix.ncentres, 1)*realmax);
        mix.covars = min(cdist);
        mix.covars = mix.covars + GMM_WIDTH*(mix.covars < eps);
    else
        % Just use variance of all data points averaged over all
        % dimensions
        mix.covars = mean(diag(cov(x)));
    end
case 'diag'

```

```

    for j = 1:mix.ncentres
        % Pick out data points belonging to this centre
        c = x(find(post(:, j)),:);
        diffs = c - (ones(size(c, 1), 1) * mix.centres(j, :));
        mix.covars(j, :) = sum((diffs.*diffs), 1)/size(c, 1);
        % Replace small entries by GMM_WIDTH value
        mix.covars(j, :) = mix.covars(j, :) + GMM_WIDTH.*(mix.covars(j, :)<eps);
    end
    case 'full'
        for j = 1:mix.ncentres
            % Pick out data points belonging to this centre
            c = x(find(post(:, j)),:);
            diffs = c - (ones(size(c, 1), 1) * mix.centres(j, :));
            mix.covars(:, j) = (diffs'*diffs)/(size(c, 1));
            % Add GMM_WIDTH*Identity to rank-deficient covariance matrices
            if rank(mix.covars(:, j)) < mix.nin
                mix.covars(:, j) = mix.covars(:, j) + GMM_WIDTH.*eye(mix.nin);
            end
        end
    case 'ppca'
        for j = 1:mix.ncentres
            % Pick out data points belonging to this centre
            c = x(find(post(:, j)),:);
            diffs = c - (ones(size(c, 1), 1) * mix.centres(j, :));
            [tempcovars, tempU, templambda] = ...
                ppca((diffs'*diffs)/size(c, 1), mix.pcca_dim);
            if length(templambda) ~= mix.pcca_dim
                error('Unable to extract enough components');
            else
                mix.covars(j) = tempcovars;
                mix.U(:, :, j) = tempU;
                mix.lambda(j, :) = templambda;
            end
        end
    otherwise
        error(['Unknown covariance type ', mix.covar_type]);
end
% end gmminit

function errstring = consist(model, type, inputs, outputs)
%CONSIST Check that arguments are consistent.
%
% Description
%
% ERRSTRING = CONSIST(NET, TYPE, INPUTS) takes a network data structure
% NET together with a string TYPE containing the correct network type,
% a matrix INPUTS of input vectors and checks that the data structure
% is consistent with the other arguments. An empty string is returned
% if there is no error, otherwise the string contains the relevant
% error message. If the TYPE string is empty, then any type of network
% is allowed.
%
% ERRSTRING = CONSIST(NET, TYPE) takes a network data structure NET
% together with a string TYPE containing the correct network type, and
% checks that the two types match.
%
% ERRSTRING = CONSIST(NET, TYPE, INPUTS, OUTPUTS) also checks that the
% network has the correct number of outputs, and that the number of
% patterns in the INPUTS and OUTPUTS is the same. The fields in NET
% that are used are
%   type
%   nin
%   nout
%
% See also
% MLPFWD
%
% Copyright (c) Ian T Nabney (1996-2001)

% Assume that all is OK as default
errstring = '';

% If type string is not empty
if ~isempty(type)
    % First check that model has type field

```

```

    if ~isfield(model, 'type')
        errstring = 'Data structure does not contain type field';
        return
    end
    % Check that model has the correct type
    s = model.type;
    if strcmp(s, type)
        errstring = ['Model type ', s, ' does not match expected type ',...
type, ''];
        return
    end
end

% If inputs are present, check that they have correct dimension
if nargin > 2
    if ~isfield(model, 'nin')
        errstring = 'Data structure does not contain nin field';
        return
    end

    data_nin = size(inputs, 2);
    if model.nin ~= data_nin
        errstring = ['Dimension of inputs ', num2str(data_nin), ...
' does not match number of model inputs ', num2str(model.nin)];
        return
    end
end

% If outputs are present, check that they have correct dimension
if nargin > 3
    if ~isfield(model, 'nout')
        errstring = 'Data structure does not contain nout field';
        return
    end
    data_nout = size(outputs, 2);
    if model.nout ~= data_nout
        errstring = ['Dimension of outputs ', num2str(data_nout), ...
' does not match number of model outputs ', num2str(model.nout)];
        return
    end
end

% Also check that number of data points in inputs and outputs is the same
num_in = size(inputs, 1);
num_out = size(outputs, 1);
if num_in ~= num_out
    errstring = ['Number of input patterns ', num2str(num_in), ...
' does not match number of output patterns ', num2str(num_out)];
    return
end
end
% end consist

function [centres, options, post, errlog] = kmeans(centres, data, options)
%KMEANS Trains a k means cluster model.
%
% Description
% CENTRES = KMEANS(CENTRES, DATA, OPTIONS) uses the batch K-means
% algorithm to set the centres of a cluster model. The matrix DATA
% represents the data which is being clustered, with each row
% corresponding to a vector. The sum of squares error function is used.
% The point at which a local minimum is achieved is returned as
% CENTRES. The error value at that point is returned in OPTIONS(8).
%
% [CENTRES, OPTIONS, POST, ERRLOG] = KMEANS(CENTRES, DATA, OPTIONS)
% also returns the cluster number (in a one-of-N encoding) for each
% data point in POST and a log of the error values after each cycle in
% ERRLOG. The optional parameters have the following
% interpretations.
%
% OPTIONS(1) is set to 1 to display error values; also logs error
% values in the return argument ERRLOG. If OPTIONS(1) is set to 0, then
% only warning messages are displayed. If OPTIONS(1) is -1, then
% nothing is displayed.
%
% OPTIONS(2) is a measure of the absolute precision required for the
% value of CENTRES at the solution. If the absolute difference between

```

```

% the values of CENTRES between two successive steps is less than
% OPTIONS(2), then this condition is satisfied.
%
% OPTIONS(3) is a measure of the precision required of the error
% function at the solution. If the absolute difference between the
% error functions between two successive steps is less than OPTIONS(3),
% then this condition is satisfied. Both this and the previous
% condition must be satisfied for termination.
%
% OPTIONS(14) is the maximum number of iterations; default 100.
%
% See also
% GMMINIT, GMMEM
%
% Copyright (c) Ian T Nabney (1996-2001)

[ndata, data_dim] = size(data);
[ncentres, dim] = size(centres);

if dim ~= data_dim
    error('Data dimension does not match dimension of centres')
end

if (ncentres > ndata)
    error('More centres than data')
end

% Sort out the options
if (options(14))
    niters = options(14);
else
    niters = 100;
end

store = 0;
if (nargout > 3)
    store = 1;
    errlog = zeros(1, niters);
end

% Check if centres and posteriors need to be initialised from data
if (options(5) == 1)
    % Do the initialisation
    perm = randperm(ndata);
    perm = perm(1:ncentres);

    % Assign first ncentres (permuted) data points as centres
    centres = data(perm, :);
end
% Matrix to make unit vectors easy to construct
id = eye(ncentres);

% Main loop of algorithm
for n = 1:niters

    % Save old centres to check for termination
    old_centres = centres;

    % Calculate posteriors based on existing centres
    d2 = dist2(data, centres);
    % Assign each point to nearest centre
    [minvals, index] = min(d2', [], 1);
    post = id(index,:);

    num_points = sum(post, 1);
    % Adjust the centres based on new posteriors
    for j = 1:ncentres
        if (num_points(j) > 0)
            centres(j,:) = sum(data(find(post(:,j)),:), 1)/num_points(j);
        end
    end

    % Error value is total squared distance from cluster centres
    e = sum(minvals);
    if store

```



```

    errlog(n) = e;
end
if options(1) > 0
    fprintf(1, 'Cycle %4d Error %11.6f\n', n, e);
end

if n > 1
    % Test for termination
    if max(max(abs(centres - old_centres))) < options(2) & ...
        abs(old_e - e) < options(3)
        options(8) = e;
        return;
    end
end
old_e = e;
end

% If we get here, then we haven't terminated in the given number of
% iterations.
options(8) = e;
if (options(1) >= 0)
    disp('Warning: Maximum number of iterations has been exceeded');
end
% end kmeans

function n2 = dist2(x, c)
%DIST2 Calculates squared distance between two sets of points.
%
% Description
% D = DIST2(X, C) takes two matrices of vectors and calculates the
% squared Euclidean distance between them. Both matrices must be of
% the same column dimension. If X has M rows and N columns, and C has
% L rows and N columns, then the result has M rows and L columns. The
% I, Jth entry is the squared distance from the Ith row of X to the
% Jth row of C.
%
% See also
% GMMACTIV, KMEANS, RBFFWD
%

% Copyright (c) Ian T Nabney (1996-2001)

[ndata, dimx] = size(x);
[ncentres, dimc] = size(c);
if dimx ~= dimc
    error('Data dimension does not match dimension of centres')
end

n2 = (ones(ncentres, 1) * sum((x.^2)', 1))' + ...
    ones(ndata, 1) * sum((c.^2)', 1) - ...
    2.*(x*(c'));

% Rounding errors occasionally cause negative entries in n2
if any(any(n2<0))
    n2(n2<0) = 0;
end
% end dist2

function [mix, options, errlog] = gmmem(mix, x, options)
%GMMEM EM algorithm for Gaussian mixture model.
%
% Description
% [MIX, OPTIONS, ERRLOG] = GMMEM(MIX, X, OPTIONS) uses the Expectation
% Maximization algorithm of Dempster et al. to estimate the parameters
% of a Gaussian mixture model defined by a data structure MIX. The
% matrix X represents the data whose expectation is maximized, with
% each row corresponding to a vector. The optional parameters have
% the following interpretations.
%
% OPTIONS(1) is set to 1 to display error values; also logs error
% values in the return argument ERRLOG. If OPTIONS(1) is set to 0, then
% only warning messages are displayed. If OPTIONS(1) is -1, then
% nothing is displayed.
%
% OPTIONS(3) is a measure of the absolute precision required of the
% error function at the solution. If the change in log likelihood

```

```

% between two steps of the EM algorithm is less than this value, then
% the function terminates.
%
% OPTIONS(5) is set to 1 if a covariance matrix is reset to its
% original value when any of its singular values are too small (less
% than MIN_COVAR which has the value eps). With the default value of
% 0 no action is taken.
%
% OPTIONS(14) is the maximum number of iterations; default 100.
%
% The optional return value OPTIONS contains the final error value
% (i.e. data log likelihood) in OPTIONS(8).
%
% See also
% GMM, GMMINIT
%

% Copyright (c) Ian T Nabney (1996-2001)

% Check that inputs are consistent
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end

[ndata, xdim] = size(x);

% Sort out the options
if (options(14))
    niters = options(14);
else
    niters = 100;
end

display = options(1);
store = 0;
if (nargout > 2)
    store = 1; % Store the error values to return them
    errlog = zeros(1, niters);
end
test = 0;
if options(3) > 0.0
    test = 1; % Test log likelihood for termination
end

check_covars = 0;
if options(5) >= 1
    if display >= 0
        %disp('check_covars is on');
    end
    check_covars = 1; % Ensure that covariances don't collapse
    MIN_COVAR = eps; % Minimum singular value of covariance matrix
    init_covars = mix.covars;
end

% Main loop of algorithm
for n = 1:niters

    % Calculate posteriors based on old parameters
    [post, act] = gmmpost(mix, x);

    % Calculate error value if needed
    if (display | store | test)
        prob = act*(mix.priors)';
        % Error value is negative log likelihood of data
        e = - sum(log(prob));
        if store
            errlog(n) = e;
        end
        if display > 0
            fprintf(1, 'Cycle %4d Error %11.6f\n', n, e);
        end
        if test
            if (n > 1 & abs(e - eold) < options(3))
                options(8) = e;
                return;
            end
        end
    end
end

```

```

        else
            eold = e;
        end
    end
end

% Adjust the new estimates for the parameters
new_pr = sum(post, 1);
new_c = post' * x;

% Now move new estimates to old parameter vectors
mix.priors = new_pr ./ ndata;

mix.centres = new_c ./ (new_pr' * ones(1, mix.nin));

switch mix.covar_type
case 'spherical'
    n2 = dist2(x, mix.centres);
    for j = 1:mix.ncentres
        v(j) = (post(:,j)'*n2(:,j));
    end
    mix.covars = ((v./new_pr))./mix.nin;
    if check_covars
        % Ensure that no covariance is too small
        for j = 1:mix.ncentres
            if mix.covars(j) < MIN_COVAR
                mix.covars(j) = init_covars(j);
            end
        end
    end
case 'diag'
    for j = 1:mix.ncentres
        diffs = x - (ones(ndata, 1) * mix.centres(j,:));
        mix.covars(j,:) = sum((diffs.*diffs).*(post(:,j)*ones(1, ...
            mix.nin)), 1)./new_pr(j);
    end
    if check_covars
        % Ensure that no covariance is too small
        for j = 1:mix.ncentres
            if min(mix.covars(j,:)) < MIN_COVAR
                mix.covars(j,:) = init_covars(j,:);
            end
        end
    end
case 'full'
    for j = 1:mix.ncentres
        diffs = x - (ones(ndata, 1) * mix.centres(j,:));
        diffs = diffs.*(sqrt(post(:,j))*ones(1, mix.nin));
        mix.covars(:,j) = (diffs'*diffs)/new_pr(j);
    end
    if check_covars
        % Ensure that no covariance is too small
        for j = 1:mix.ncentres
            if min(svd(mix.covars(:,j))) < MIN_COVAR
                mix.covars(:,j) = init_covars(:,j);
            end
        end
    end
case 'ppca'
    for j = 1:mix.ncentres
        diffs = x - (ones(ndata, 1) * mix.centres(j,:));
        diffs = diffs.*(sqrt(post(:,j))*ones(1, mix.nin));
        [tempcovars, tempU, templambda] = ...
            ppca((diffs'*diffs)/new_pr(j), mix.pcca_dim);
        if length(templambda) ~= mix.pcca_dim
            error('Unable to extract enough components');
        else
            mix.covars(j) = tempcovars;
            mix.U(:, j) = tempU;
            mix.lambda(j, :) = templambda;
        end
    end
    if check_covars
        if mix.covars(j) < MIN_COVAR
            mix.covars(j) = init_covars(j);
        end
    end
end

```

```

        end
        otherwise
            error(['Unknown covariance type ', mix.covar_type]);
        end
    end
end

options(8) = -sum(log(gmmprob(mix, x)));
if (display >= 0)
    disp('Warning: Maximum number of iterations has been exceeded');
end
% end gmmem

function [post, a] = gmmpost(mix, x)
%GMMPOST Computes the class posterior probabilities of a Gaussian mixture model.
%
% Description
% This function computes the posteriors POST (i.e. the probability of
% each component conditioned on the data  $P(J|X)$ ) for a Gaussian mixture
% model. The data structure MIX defines the mixture model, while the
% matrix X contains the data vectors. Each row of X represents a
% single vector.
%
% See also
% GMM, GMMACTIV, GMMPROB
%
% Copyright (c) Ian T Nabney (1996-2001)

% Check that inputs are consistent
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end

ndata = size(x, 1);

a = gmmactiv(mix, x);

post = (ones(ndata, 1)*mix.priors).*a;
s = sum(post, 2);
% Set any zeros to one before dividing
s = s + (s==0);
post = post./(s*ones(1, mix.ncentres));

% end gmmpost

function a = gmmactiv(mix, x)
%GMMACTIV Computes the activations of a Gaussian mixture model.
%
% Description
% This function computes the activations A (i.e. the probability
%  $P(X|J)$  of the data conditioned on each component density) for a
% Gaussian mixture model. For the PPCA model, each activation is the
% conditional probability of X given that it is generated by the
% component subspace. The data structure MIX defines the mixture model,
% while the matrix X contains the data vectors. Each row of X
% represents a single vector.
%
% See also
% GMM, GMMPOST, GMMPROB
%
% Copyright (c) Ian T Nabney (1996-2001)

% Check that inputs are consistent
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end

ndata = size(x, 1);
a = zeros(ndata, mix.ncentres); % Preallocate matrix

switch mix.covar_type
case 'spherical'

```

```

% Calculate squared norm matrix, of dimension (ndata, ncentres)
n2 = dist2(x, mix.centres);

% Calculate width factors
wi2 = ones(ndata, 1) * (2 .* mix.covars);
normal = (pi .* wi2) .^ (mix.nin/2);

% Now compute the activations
a = exp(-(n2./wi2))./ normal;

case 'diag'
    normal = (2*pi)^(mix.nin/2);
    s = prod(sqrt(mix.covars), 2);
    for j = 1:mix.ncentres
        diffs = x - (ones(ndata, 1) * mix.centres(j, :));
        a(:, j) = exp(-0.5*sum((diffs.*diffs)./(ones(ndata, 1) * ...
            mix.covars(j, :)), 2)) ./ (normal*s(j));
    end

case 'full'
    normal = (2*pi)^(mix.nin/2);
    for j = 1:mix.ncentres
        diffs = x - (ones(ndata, 1) * mix.centres(j, :));
        % Use Cholesky decomposition of covariance matrix to speed computation
        c = chol(mix.covars(:, :, j));
        temp = diffs/c;
        a(:, j) = exp(-0.5*sum(temp.*temp, 2))./(normal*prod(diag(c)));
    end

case 'ppca'
    log_normal = mix.nin*log(2*pi);
    d2 = zeros(ndata, mix.ncentres);
    logZ = zeros(1, mix.ncentres);
    for i = 1:mix.ncentres
        k = 1 - mix.covars(i)./mix.lambda(i, :);
        logZ(i) = log_normal + mix.nin*log(mix.covars(i)) - ...
            sum(log(1 - k));
        diffs = x - ones(ndata, 1)*mix.centres(i, :);
        proj = diffs*mix.U(:, :, i);
        d2(:, i) = (sum(diffs.*diffs, 2) - ...
            sum((proj.*(ones(ndata, 1)*k)).*proj, 2)) / ...
            mix.covars(i);
    end
    a = exp(-0.5*(d2 + ones(ndata, 1)*logZ));
otherwise
    error(['Unknown covariance type ', mix.covar_type]);
end

% end gmmactiv

function prob = gmmprob(mix, x)
%GMMPROB Computes the data probability for a Gaussian mixture model.
%
% Description
% This function computes the unconditional data density P(X) for a
% Gaussian mixture model. The data structure MIX defines the mixture
% model, while the matrix X contains the data vectors. Each row of X
% represents a single vector.
%
% See also
% GMM, GMMPOST, GMMACTIV
%
% Copyright (c) Ian T Nabney (1996-2001)

% Check that inputs are consistent
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end

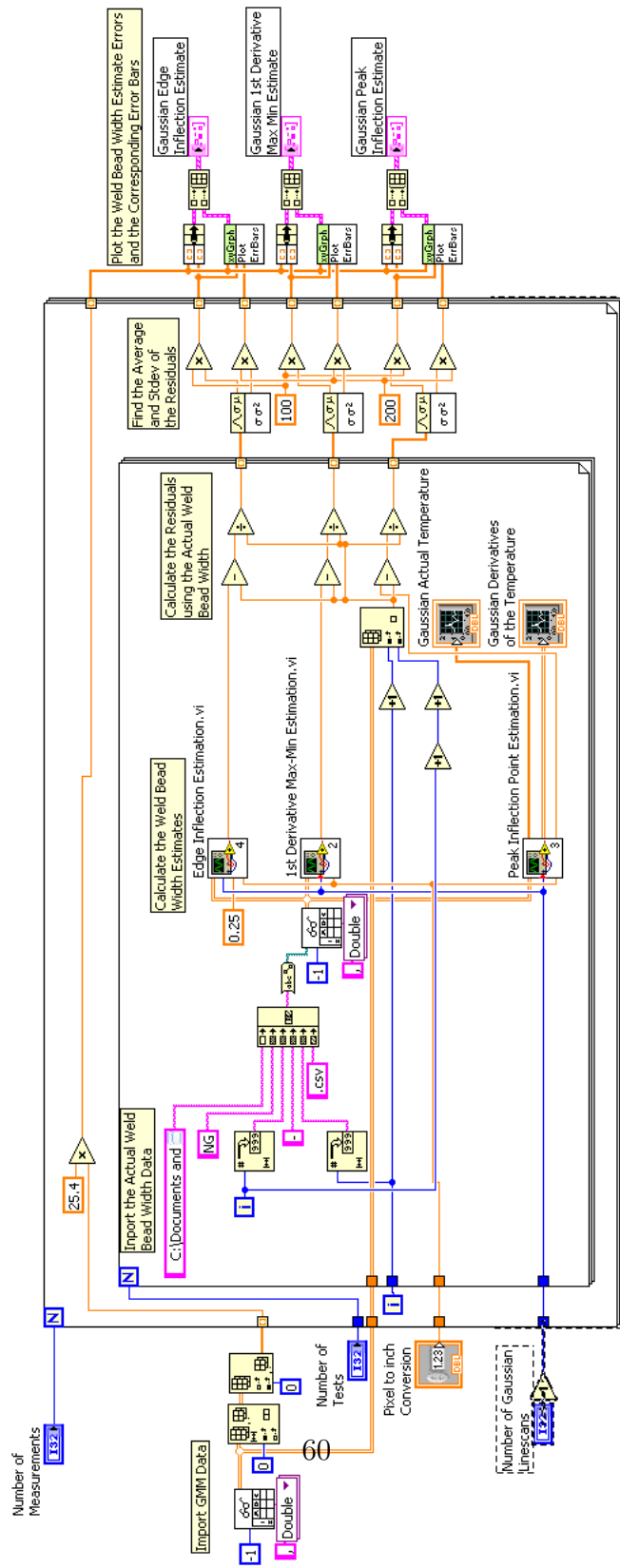
% Compute activations
a = gmmactiv(mix, x);

% Form dot product with priors
prob = a * (mix.priors)';

```

```
% end gmmprob
```

B.2 Weld Bead Width Estimation using GMM Fits



Bibliography

- [1] B. Venkatraman et al. Thermography for online detection of incomplete penetration and penetration depth estimates. *Asia-Pacific Conference on NDT*, 603(102), 2006.
- [2] M. Menaka et al. Estimating bead width and depth of penetration during welding by infrared thermal imaging. *Insight*, 47(9):564–568, 2005.
- [3] P. Banerjee et al. Infrared sensing for on-line weld geometry monitoring and control. *Journal of Engineering for Industry*, 117:323–330, 1995.
- [4] S. Nagarajan et al. Thermal image analysis techniques for seam tracking in arc welding. *SPIE Infrared Technology XIV*, 972:256–267, 1988.
- [5] W. H. Chen et al. Estimating bead width and depth of penetration during welding by infrared thermal imaging. *SPIE Infrared Technology*, 972:268–272, 1988.

Vita

Patrick John William Casey was born in Flagstaff, Arizona on January 23, 1985 to Jack and Gretchen Casey. After completing his work at McQueen High School in Reno, Nevada in 2003 he received his Bachelor of Science in Mechanical Engineering at the University of Nevada, Reno in May 2007. In August 2007, he entered the Graduate School of The University of Texas at Austin.

Permanent address: 3359 Snake River Dr.
Reno, NV 89503

This thesis was typeset with L^AT_EX[†] by the author.

[†]L^AT_EX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's T_EX Program.